INFORMATIQUE
TECHNIQUE

Expert

Développer des services REST en Java

Échanger des données au format JSON

Téléchargement
www.editions-eni.fr

Aurélie SOBRERO

eni Editions

Table des matières

Les éléments à télécharger sont disponibles à l'adresse suivante :
http://www.editions-eni.fr
Saisissez la référence ENI de l'ouvrage **EIREST** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

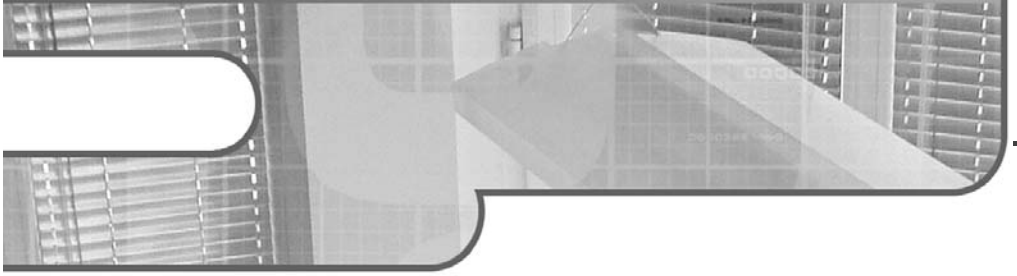
Chapitre 3
Introduction à REST

Chapitre 4
Production de JSON avec Java et REST

Chapitre 5
Pour aller plus loin

Chapitre 6
Exceptions communes

Webographie



Chapitre 4
Production de JSON avec Java et REST

1. Du POJO au JSON grâce aux annotations JAX-RS

JAX-RS est l'acronyme pour *Java API for RESTFul Services*.

Cette API (*Application Programming Interface*) met à disposition des annotations pour créer simplement des services REST.

1.1 Les bases

Une fois choisie et ajoutée dans votre projet, l'implémentation de JAX-RS, la première étape est de choisir la méthode HTTP qui appellera telle ou telle méthode de votre application.

1.1.1 Choix de la méthode HTTP

Bien que le protocole HTTP soit large, comme vu dans le chapitre Introduction à REST, les méthodes prises en charge dans JAX-RS se limitent aux méthodes OPTIONS, HEAD, GET, POST, PUT et DELETE. Vous ne pourrez donc pas utiliser PATCH, LINK,UNLINK, TRACE et CONNECT.

Les annotations suivantes sont disponibles :

– @OPTIONS pour connaître les méthodes implémentées.
– @HEAD pour connaître les en-têtes d'une ressource.
– @GET pour récupérer une ressource.
– @POST pour créer une ressource.
– @PUT pour modifier une ressource.
– @DELETE pour supprimer une ressource.

Ces annotations sont à positionner dans la classe, non pas à la déclaration de la classe, mais à la déclaration de la méthode qui sera exécutée.

@HEAD

Il est à noter qu'il n'est pas obligatoire d'implémenter la méthode HEAD. Pour répondre à une requête HEAD envoyée au service REST, JAX-RS regarde s'il y a une méthode avec l'annotation @HEAD, et si la méthode n'existe pas, c'est la méthode avec l'annotation @GET qui est automatiquement exécutée, mais son corps ne sera pas renvoyé. Cependant, les performances peuvent être moindres dans ce cas.

@OPTIONS

Comme pour l'annotation @HEAD, il est possible mais non obligatoire d'implémenter manuellement la méthode OPTIONS à l'aide de l'annotation @OPTIONS. Si la méthode n'existe pas, le résultat sera généré automatiquement à l'aide des méthodes définies dans la classe appelée.

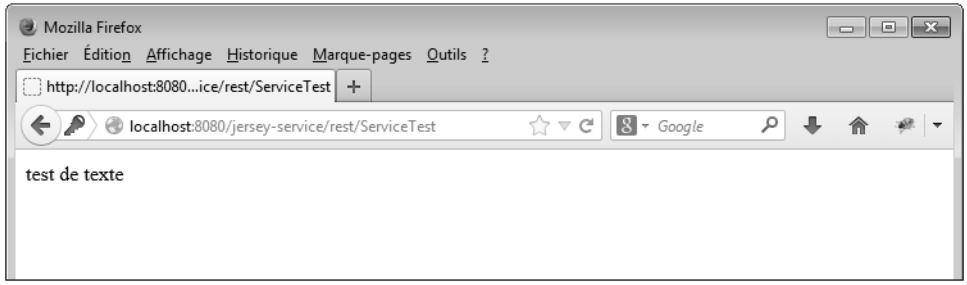
1.1.2 Choix du chemin d'appel

Une fois votre méthode choisie, il reste à déterminer l'URL qui sera appelée. Elle sera relative à votre application, il ne faut donc pas remettre ni le protocole, ni le nom de domaine, ni le chemin de contexte de l'application. Cela passe par l'utilisation de l'annotation @Path qui prend en paramètre le chemin d'appel.

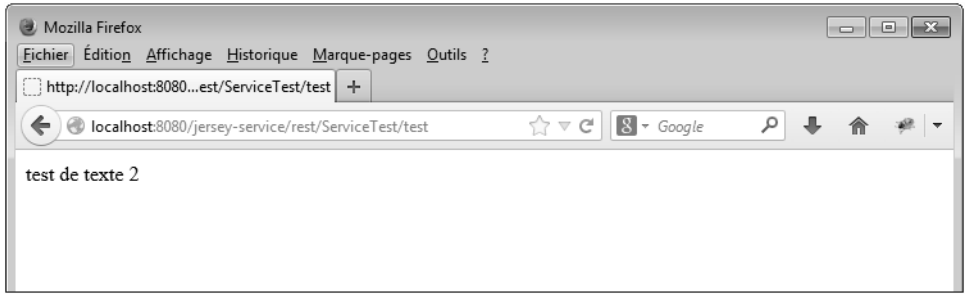
Cette annotation se positionne en deux endroits : obligatoirement à la déclaration de la classe, et de manière facultative à la déclaration de la méthode qui sera exécutée.

```
@Path("/ServiceTest")
public class ServiceTest {

    @GET
    public String test() {
        return "test de texte";
    }

    @GET
    @Path("/test")
    public String test2() {
        return "test de texte 2";
    }
}
```

L'appel de /ServiceTest provoquera la sortie textuelle suivante.



L'appel de /ServiceTest/test provoquera la sortie textuelle suivante.



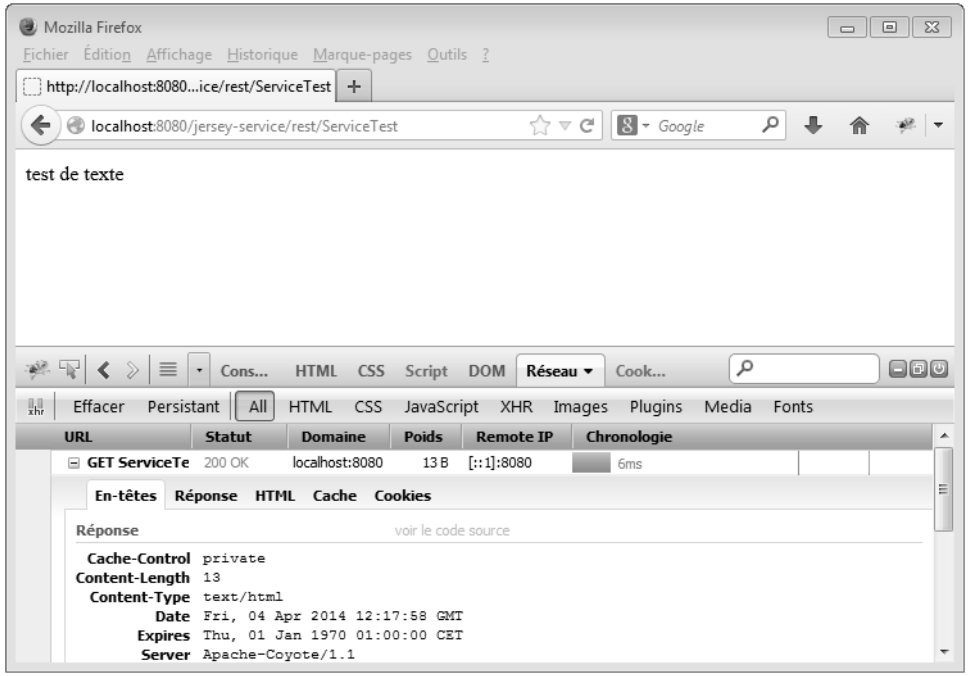
1.1.3 Choix du type de retour

Après avoir choisi la méthode HTTP et l'URL relative d'appel, il faut choisir le type MIME de sortie.

Pour ce faire, il faut utiliser l'annotation `@Produces`, avec un attribut de la classe `MediaType` en paramètre.

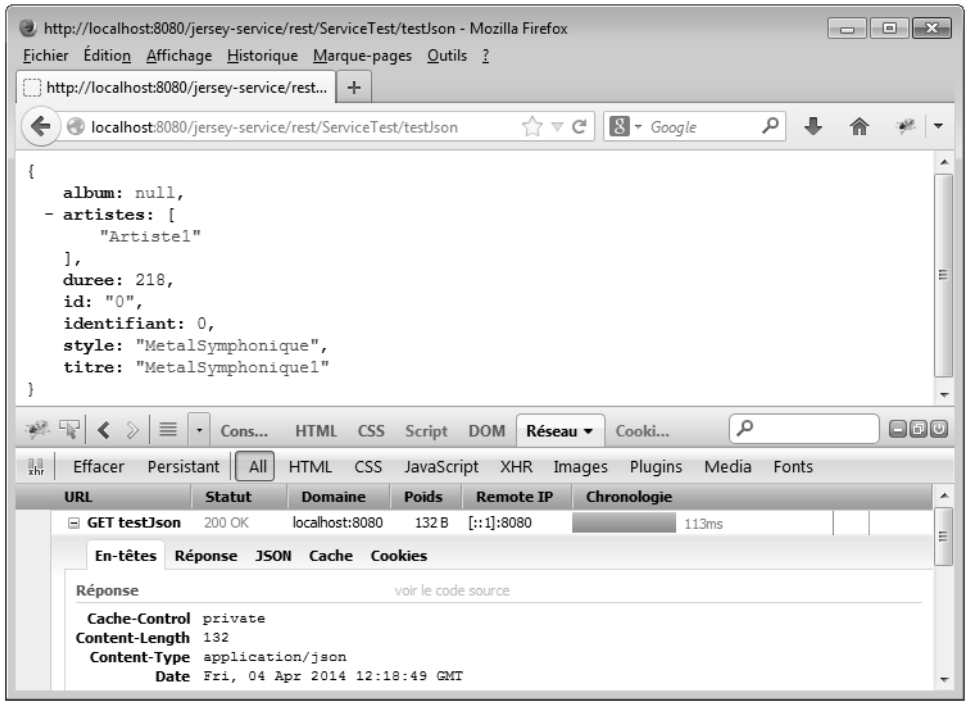
Par défaut, si l'annotation n'est pas placée, du **text/html** sera généré.

```
@Path("/ServiceTest")
public class ServiceTest {

    @GET
    public String test() {
        return "test de texte";
    }
}
```



Dans l'exemple ci-après, le retour sera catégorisé JSON, grâce à la présence d'un argument, qui peut être une chaîne de caractères manuelle ou l'un des types prédéfinis dans la classe `MediaType`.

```
@Path("/ServiceTest")
public class ServiceTest {

    [...]

    @GET
    @Path("/testJson")
    @Produces(MediaType.APPLICATION_JSON)
    public String jsonTest() {
        final TitreMusique monTitreMusique
            = new TitreMusique("MetalSymphonique1", 218F);

        monTitreMusique.setStyle("MetalSymphonique");
        monTitreMusique.ajouteArtiste("Artiste1");

        final JSON json = new JSON(2);
        return json.format(monTitreMusique);
    }
}
```



Il est possible de mettre plusieurs arguments, en les listant entre accolades. Le retour dépendra alors du type demandé par l'application appelante, grâce à l'en-tête `HTTP Accept`.

```
@Path("/ServiceTest")
public class ServiceTest {

    [...]

    @GET
    @Path("/testMultiple")
    @Produces({
        MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML
    })
    public String testMultiple() {
        final TitreMusique monTitreMusique
            = new TitreMusique("MetalSymphonique1", 218F);

        monTitreMusique.setStyle("MetalSymphonique");
```